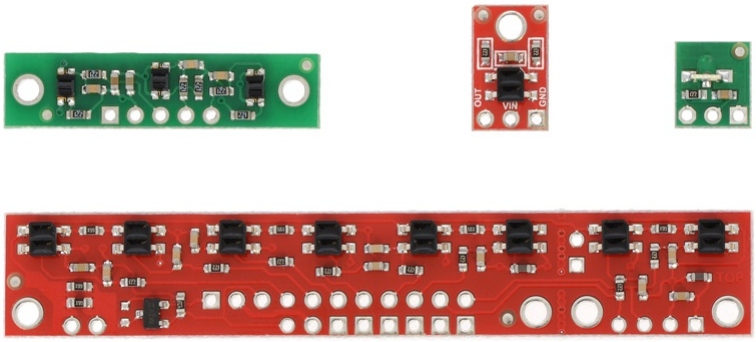


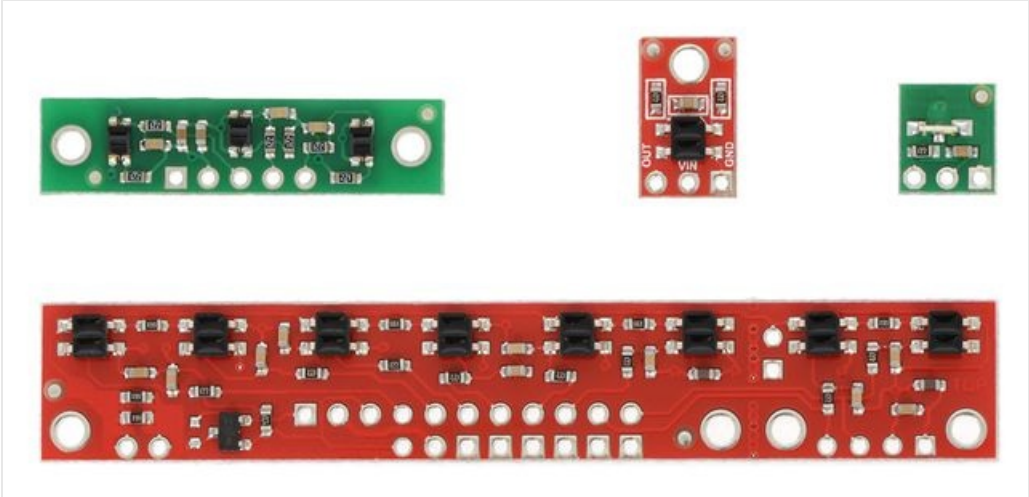
Guide utilisateur du senseur QTR (suiveur de ligne)

Un guide utiliser et exploiter un senseur QTR (détecteur de ligne).
(version 0.1)

Traduit par MicroControleur Hobby (shop.mchobby.be)
Compilé depuis la traduction maintenue sur https://wiki.mchobby.be/index.php?title=Pololu-Senseur-QTR
Les hyperliens sont disponibles sur la version en ligne du document.
Translated by MicroControleur Hobby (shop.mchobby.be)
Compiled from online translation available at https://wiki.mchobby.be/index.php?title=Pololu-Senseur-QTR
Hyperlinks are available on the online version of this document.

Introduction

Introduction



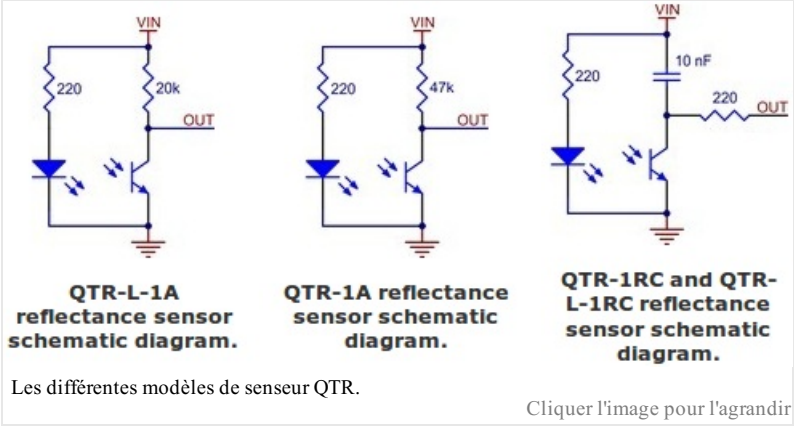
Senseur QTR, comparaison des tailles. Dans le sens horaire depuis en haut à gauche: QTR-3RC, QTR-1RC, QTR-L-1RC, QTR-8RC.

Cliquer l'image pour l'agrandir

Le senseur réflectif QTR de Pololu embarque des paires de LED infrarouge et phototransistor utilisé pour réaliser une mesure analogique de la réflexion IR (InfraRouge), ce qui en fait un excellent candidat pour la détection de bord (proximité-proche) et détection de ligne. Les modules sont compacts en vendu en unité simple (QTR-1A et QTR-1RC; QTR-L-1A et QTR-L-1RC), assemblage de 3-senseurs (des *arrays* en anglais, comme le QTR-3A et QTR-3RC) ou en assemblage de 8 senseurs (QTR-8A et QTR-8RC) qui peut optionnellement être divisé en deux sections (deux *arrays*) de respectivement 2-senseurs et 6-senseurs.

Les modules sont également disponible avec différentes interface de sorties:

- Le QTR-x**A** a une sortie en tension analogique entre 0 et Vcc. Tension qui peut être relevée avec un convertisseur analogique-vers-digital (ADC)
- Le QTR-xR**C** qui utilise une sortie digital. Ce senseur requière un microcontrôleur capable de placer un entrée au niveau haut et qui mesure le temps nécessaire à la tension pour chuter au niveau bas. Cette mesure de temps correspond à une lecture *analogique* de la réflectance https://fr.wikipedia.org/wiki/R%C3%A9flectance (*wikipedia*).



Les différentes modèles de senseur QTR.

Cliquer l'image pour l'agrandir

Ce document explique comment installer la bibliothèque Arduino pour les senseur QTR de réflectance. La bibliothèque offre également un croquis d'exemple ainsi que des liens vers la documentation de la bibliothèque. La bibliothèque contient tout ce dont vous aurez besoin pour interfacer le senseur de réflectance QTR-8x ou de multiple senseur de réflectance QTR-1x, incluant des fonctionnalités avancées comme la calibration automatique, la détection de ligne et calcul de la position de la ligne.

A Propos de MCHobby
Crédit de traduction

Sommaire

A propose de .: MC Hobby :.

MCHobby investit du temps et de l'argent dans la réalisation de traduction et/ou documentation. C'est un travail long et fastidieux réalisé dans l'esprit Open-Source... donc gratuit et librement accessible.
SI vous aimez nos traductions et documentations ALORS aidez nous à en produire plus en achetant vos produits chez MCHobby https://shop.mchobby.be .

MC Hobby SPRL, basé en Belgique, est le traducteur de ce manuel.

MC Hobby cherche à promouvoir, en français, la plateforme Open-Source Arduino, ses extensions et exemples de programmation afin de la rendre accessible au plus grand nombre.

License

Crédit de traduction

Toute référence, mention ou extrait de cette traduction doit être explicitement accompagné du texte de crédit de traduction.

Ce texte est repris en "pied de document" sur toutes les pages/documents, merci d'en prendre connaissance.

Limite de traduction

Anglicisme

Du fait que de nombreux anglicismes soient utilisés au jour le jour dans les milieux techniques, je me suis permis d'en préserver quelques-uns qui me semblaient avoir plus de sens dans la langue de Shakespeare que traduit dans la langue de Voltaire. Par ailleurs, ces anglicismes pourraient se révéler fort utiles lors de vos prochaines recherches sur Internet.

Ainsi, vous retrouverez les termes suivants :

- Pin : fait référence à une « broche » de raccordement d'un composant (Anglicisme fort répandu).
- Datasheet : fait référence à la « fiche technique » d'un composant.
- Pin Header : malheureusement intraduisible sans en perdre le sens mais vous aurez vite identifié les « Pinheaders ». La traduction courant est *connecteur*, ce terme restant beaucoup trop vague.
- LED : ce sont les diodes électroluminescentes (aussi appelées DEL). Cependant, le terme LED est si répandu qu'il a été préservé.

- RGB : Fait référence aux 3 couleurs fondamentales Rouge Vert Bleu (Red Green Blue). Si le terme RVB est d'usage fréquent en Belgique et en France, l'acronyme RGB reste encore le plus répandu.

Code source

Nous avons traduit les commentaires dans les programmes afin de les rendre plus intelligibles.

Par contre, nous n'avons pas modifié les codes sources (nom des variables, ...) qui, eux, restent ceux fournit par le concepteur et le fabricant du kit.

Autorisations

Le présent manuel a été traduit avec l'autorisation de Pololu (www.pololu.com).

Signaler une erreur

Malgré tout le soin apporté à la réalisation de cette traduction, il n'est pas impossible qu'une erreur se soit malgré tout glissée dans ce document. N'hésitez pas à nous envoyez un e-mail si vous en constatiez une. Pour adresser vos remarques et commentaires relatifs à la traduction, ou pour demander une traduction complémentaire, contactez nous par e-mail à

- support (arobase) mchobby.be.

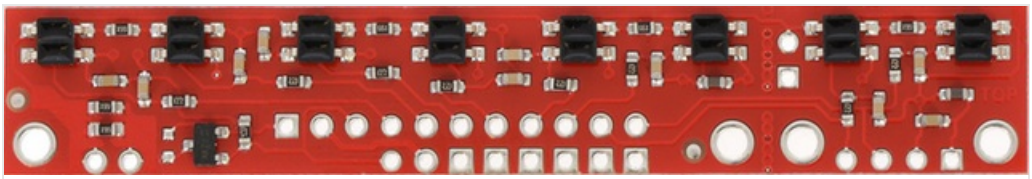
N'oubliez pas de mentionner le manuel/page/lien en mentionnant l'erreur ;-)

Principe de fonctionnement

Sommaire

Le senseur de ligne

L'ensemble des senseurs du QTR-8RC (détection par réfléchissement) sont utilisés comme détecteur de ligne mais peuvent également servir comme senseur de proximité ou senseur de surface réfléchissante.



Module QTR8C de pololu.

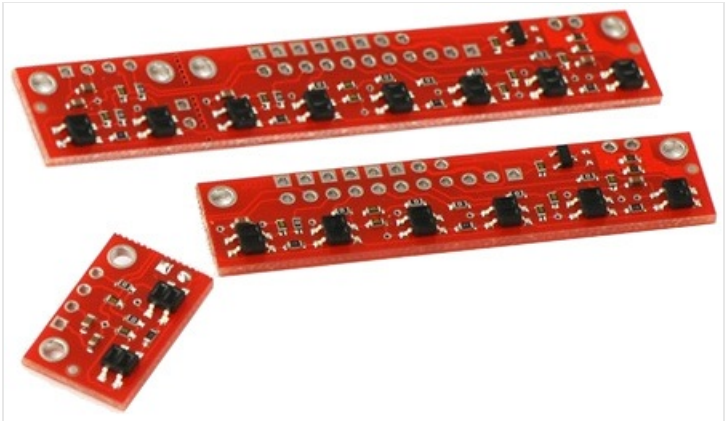
Cliquer l'image pour l'agrandir

Le module QTR8C https://shop.mchobby.be/product.php?id_product=497 lien pololu https://www.pololu.com/product/961 est équipé de 8 émetteurs infrarouges et 8 senseurs infrarouges (des phototransistors) montés en paires et espacés de 9.525 mm.

Ce senseur est conçu pour que la carte soit placée parallèlement à la surface à surveiller.

Séparable en deux

Si vous n'avez pas besoin des 8 senseurs (ou s'il n'y a pas assez de place) alors il est possible de scinder la section en deux!



Module QTR8C séparable.

Cliquer l'image pour l'agrandir

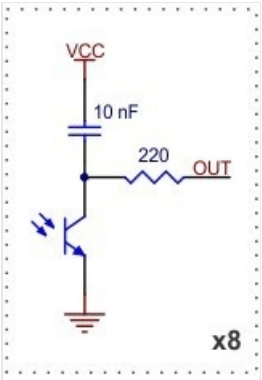
De la sorte, il est possible de disposer d'une section à 6 senseurs et une autre section de 2 senseurs (comme visible ci-dessus).

La carte peut être incisée sur les deux côtes (le long des perforations) et ensuite être courbé jusqu'à ce qu'il soit scindé. Chacune des deux pièces résultante peut être utilisé comme un senseur indépendant.

La section à 2 senseurs nécessite le placement d'une résistance complémentaire pour limiter le courant des LEDs (résistance incluse dans le kit).

Module QTR8C séparé.
Cliquer l'image pour l'agrandir

Les photo-transistors



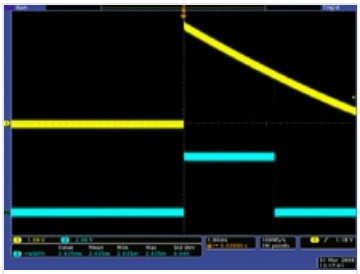
QTRxC principe de fonctionnement.
Cliquer l'image pour l'agrandir

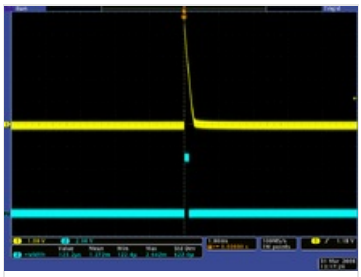
Pour utiliser le senseur, il faut d'abord activer la broche de lecture du microcontrôleur en sortie. Configurée en sortie, la broche permet de charger la capacité du noeud en appliquant une tension de 5V sur la broche OUT (broche de sortie du circuit).

Ensuite, on passe broche en entrée pour lire la réflectance http://fr.wikipedia.org/wiki/R%C3%A9flectance (Wikipedia.fr) en lisant la tension sur cette broche. En effet, une fois reconfigurée en entrée, la broche présente une très haute impédance et la seule opportunité de déchargement de la capacité, c'est par l'intermédiaire du phototransistor. La courant traversant le transistor dépend de son excitation... et donc directement proportionnel à la luminosité qu'il reçoit (donc de la lumière renvoyée par la surface réfléchissante).

Il suffit de surveiller la vitesse de la chute de tension pour déterminer la réflectance de la surface sous le senseur:

- La tension chutera beaucoup plus vite pour une surface blanche car le photo-transistor, plus excité, conduit plus de courant. La capacité se déchargera plus rapidement.
- La tension chutera lentement si la surface renvoie peu de lumière vers le photo-transistor. C'est le cas pour une surface opaque.


La sortie du QTR-1RC (jaune) lorsque le senseur passe au dessus d'une ligne noire (à 3.1mm) et correspondance du temps d'impulsion de cette sortie sur un microcontrôleur (bleu).
Cliquer l'image pour l'agrandir


La sortie du QTR-1RC (jaune) lorsque le senseur passe au dessus d'une surface blanche (à 3.1mm) et correspondance du temps d'impulsion de cette sortie sur un microcontrôleur (bleu).
Cliquer l'image pour l'agrandir

La tension sur la broche OUT, chutera donc plus ou moins vite. La bibliothèque Arduino doit mesurer le temps nécessaire pour que la tension chute suffisamment pour ramener la broche OUT à l'état bas. Ce temps est un bon indicateur de la lumière infrarouge renvoyée vers senseur infrarouge (photo-transistor) et donc du type de surface réfléchissante sous le senseur.

L'approche utilisée pour réaliser la mesure a plusieurs avantages, plus particulièrement avec le module QTR8C capable de désactiver ses LEDs infrarouges.

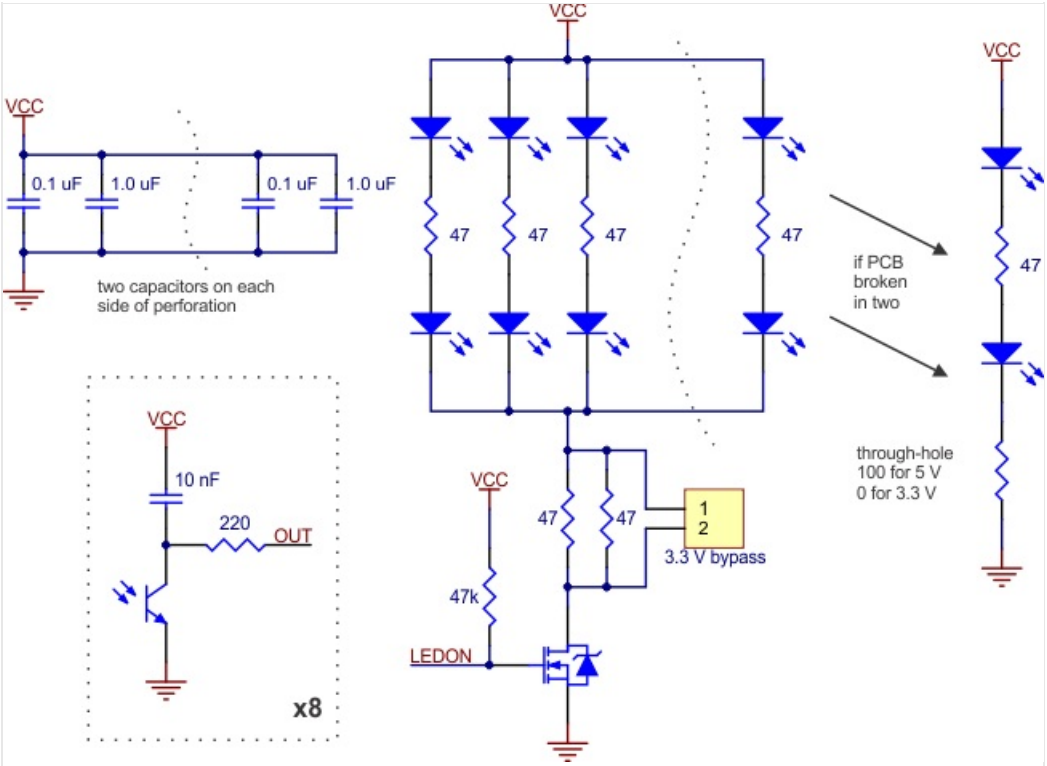
Quels sont les avantages:

- Pas besoin d'un convertisseur digital/analogique (ADC).

- Améliore la sensibilité par rapport à une sortie analogique utilisant pont diviseur de tension.
- Il est possible de lire plusieurs senseurs en même temps (sur la plupart des microcontrôleurs).
- La lecture en parallèle permet d'optimiser l'activation des LEDs et d'optimiser la consommation.

Activation des LEDs

Toutes les sorties sont indépendantes mais les LEDs sont infrarouges sont connectées en série par paires (pour diviser la consommation par deux).



Les LEDs du QTRxC

Cliquer l'image pour l'agrandir

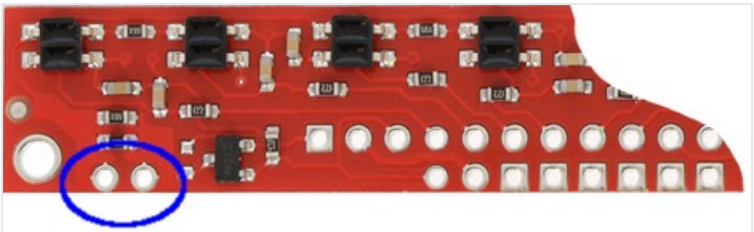
Les LEDs sont contrôlées par un MOSFET avec la gate maintenue à VCC via une résistance Pull-Up. Cela permet au microcontrôleur désactiver les LEDs en ramenant le potentiel de la gate du MOSFET à 0 volts (niveau bas).

Pouvoir désactiver les LEDs permet de limiter la puissance consommée par le projet lorsque le senseur n'est pas utilisé. Il est également possible de contrôler la luminosité de celles-ci à l'aide d'un signal PWM.

Les LEDs sont connectées par paire en série (pour diviser la consommation par deux). Les LEDs sont contrôlées par l'intermédiaire d'un MOSFET dont la gate maintenue à VCC avec une résistance pull-up. Ramener le potentiel au niveau LOW (0v) permet de désactiver les LEDs.

Sous 3.3V

Les résistances de limitation de courant des LED sont prévu pour un fonctionnement sous 5 V.



Le QTR8C sous 3.3v

Cliquer l'image pour l'agrandir

Ces résistances sont organisées en deux étages; ce qui permet de bypasser un étage pour autoriser le fonctionnement sous 3.3 V.

Consommation

Le courant d'une LED est d'approximativement 20–25 mA, ce qui fait représente un **courant de fonctionnement total de 80-100mA** pour tout le module.

Fonctionnement de la bibliothèque

Le module QTR-8RC dispose de 8 sorties identiques (le QTR-1RC à une seule sortie) et nécessite des entrées/sorties digitales capable de piloter le signal de sortie pour imposer un niveau logique HIGH. Par la suite, passer cette broche en entrée pour mesurer le temps nécessaire à la tension pour chuter.

La séquence à utiliser pour lire un senseur est la suivante:

- Allumer les LEDs Infrarouges (optionnel).
- Placer la ligne digital en sortie (mode OUTPUT) et placer la ligne au niveau HAUT (HIGH).
- Attendre au moins 10 µs pour permettre au noeud de se charger.
- Placer la ligne digital en entrée (haute impédance).
- Mesurer le temps nécessaire à la ligne pour retomber au niveau bas.
- Eteindre les LEDs Infrarouges (optionnel).

En général, ces étapes sont conduites simultanément sur plusieurs lignes d'entrées/sorties du module.

Avec un forte réflectance, le temps de décroissance de la tension peut être de quelques dizaines de **micro-secondes**; Sans aucune réflectance, le temps de décroissance de la tension peut être de plusieurs **millisecondes**.

Le temps de décroissance exacte dépends des caractéristiques des entrées/sorties de votre microcontrôleur. Plus précisément de l'impédance de la ligne lorsque celle-ci est configurée en entrée.

Pour les cas typiques, les résultats significatifs sont obtenus endéans la milliseconde (1 ms), ce qui autorise un échantillonnage des 8 senseurs à une fréquence pouvant atteindre 1 kHz (1000 fois par secondes) avec une consommation moyenne de 100mA puisque les LEDs infrarouges restent allumées.

Si un échantillonnage à basse fréquence est suffisant, il est possible d'économiser une puissance substantielle en désactivant les LEDs Infrarouges. Par exemple, un échantillonnage à 100 Hz (100 fois par secondes), les LEDs peuvent être désactivées pendant 90% du temps, ce qui ramène la consommation moyenne de 10mA (au lieu de 100mA).

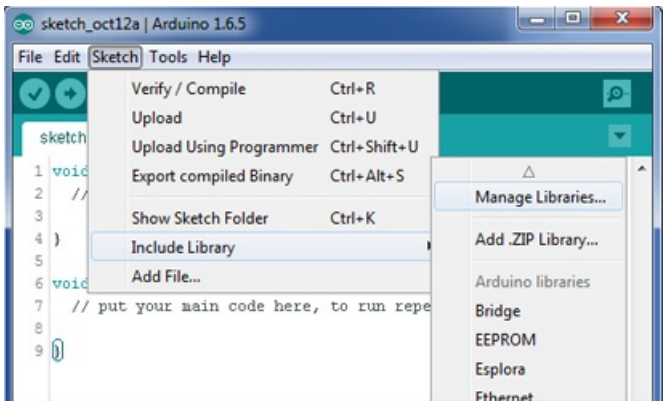
Note:

Les temps de décroissance de la tension de l'ordre de 10ms (forte réflectance) contre plusieurs ms (faible réflectance) ne permettent pas de mesurer -de façon fiable- de subtiles différences dans des environnement à basse réflectances.

Installer bibliothèque

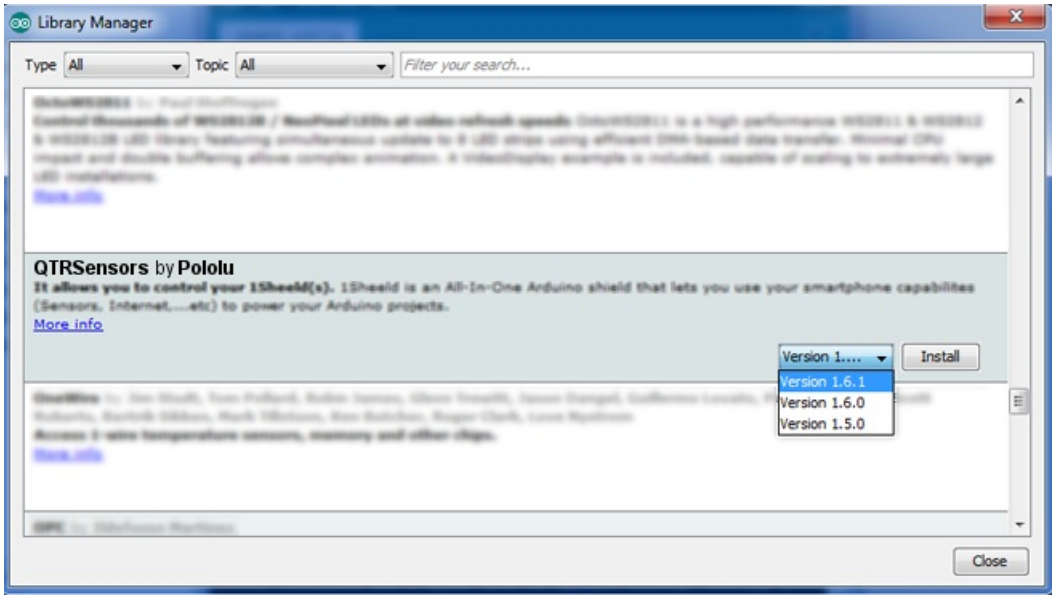
Installer la bibliothèque

Si vous utiliser la version 1.6.2 ou plus récente du logiciel Arduino https://www.arduino.cc/en/Main/Software ,vous pourrez utiliser le gestionnaire de bibliothèque pour installer la bibliothèque:



Dans Arduino IDE, ouvrez le menu "Croquis" et sélectionnez "Inclure une bibliothèque... | Gérer les bibliothèques...".

1. Cherchez après "QTRSensors".
2. Cliquez sur l'entrée "QTRSensors" affiché dans la liste.
3. Cliquez sur le bouton "Installer".



Installer à la main

Si vous ne disposez pas du gestionnaire de bibliothèque, vous pouvez l'installer manuellement.

Téléchargez QTRSensors.zip https://github.com/pololu/qtr-sensors-arduino/archive/master.zip

1. Télécharger la dernière version de l'archive depuis GitHub https://github.com/pololu/qtr-sensors-arduino/releases (voir aussi le lien ci-dessus) et la décompresser.
2. Renommer le répertoire "qtr-sensors-arduino-xxxx" vers "QTRSensors".
3. Ensuite, placer le répertoire "QTRSensors" dans le répertoire des bibliothèque (*libraries*) du répertoire contenant vos croquis arduino. Pour localiser facilement ce répertoire en sélectionnant le point de menu "Fichier | Préférences" dans Arduino IDE. S'il n'y a pas encore de sous répertoire "libraries" à cette position alors il sera nécessaire de le créer avant d'y déplacer QTRSensors.
4. Redémarrer Arduino IDE après installer la biblothèque.

Tester la bibliothèque

Un fois la bibliothèque installé, vous pouvez l'intégrer dans votre croquis en sélectionnant le point de menu "Croquis | Importer bibliothèque | QTRSensors" depuis votre Arduino IDE (ou saisissez simplement `#include <QTRSensors.h>` en haut de votre croquis).

Notez qu'il serait probablement nécessaire de redémarrer Arduino IDE avant de pouvoir coir la nouvelle bibliothèque.

Une fois fait, vous pouvez créer:

- pour un senseur QTR-xA : un objet **QTRSensorsAnalog** destiné au senseur analogique (mesure de tension).
- pour un senseur QTR-xRC : un objet **QTRSensorsRC** destiné au senseur digital (mesure de temps).

```
// Créer un objet pour 3 senseurs QTR-xA sur les
// entrées analogique 0, 2 et 6
QTRSensorsAnalog qtra((unsigned char[]) {0, 2, 6}, 3);

// Créer un objet pour senseurs QTR-xRC sur les
// entrées digitales 0 et 9 ainsi que sur les
// entrée analogiques 1 et 3 (qui sont utilisées
// en tant qu'entrée digitale 15 et 17)
QTRSensorsRC qtrrc((unsigned char[]) {0, 9, 15, 17}, 4);
```

Cette bibliothèque prend en charge -en interne- les différences entre les senseurs QTR-xA et QTR-xRC. Grâce à la bibliothèque vous disposez d'une interface commune pour les deux types de senseur. Comme vous pouvez le constater dans le code d'exemple ci-dessus, la seule différence se trouve au niveau du constructeur.

Le premier argument du constructeur `QTRSensorsAnalog` est un tableur de broche analogique (0 – 7) alors que le premier argument du constructeur `QTRSensorsRC` est un tableau de broche digital (0 – 19).

Note: les broches analogiques 0 – 5 peuvent être utilisée comme les broches digitales de 14 à 19. Voyez la page utilisation et notes pour plus de détails.

La seule différence que vous pourriez noter, c'est le temsp nécessaire pour lire les valeurs du senseur. Les senseur QTR-xRC sont tous lus en parallèle mais nécessite le temps de traitement d'une impulsion qui peut prendre jusqu'à 3ms (vous pouvez spécifier à la bibliothèque le temps max *-timeout-* de cette impulsion avant avant de retourner un résultat comme "noir"). Les senseur QTR-xA utilise le convertisseur analogique-vers-digital (ADC) et doit donc lire les entrée séquentiellement. De surcroît, le résultat analogique est produit par un calcul de moyenne sur plusieurs échantillons pour chacun des senseurs (il est possible de préciser le nombre d'échantillons). L'utilisation d'une moyenne permet d'atténuer l'effet du bruit.

La bibliothèque contient différents croquis pour vous aider à démarrer. Pour voir les croquis d'exemples, ouvrez Arduino IDE et naviguez dans le menu:

Fichier > Exemples > QTRSensors

Comme première étape, nous recommandons d'utiliser l'exemple **QTR*x*RawValuesExemple**:

- soit **QTRARawValuesExample** pour les senseurs QTR-xA.
- soit **QTRRCRawValuesExample** pour les senseurs QTR-xRC.

Ces exemples affiche simplement le résultat brute de la lecture des senseurs vers le moniteur série (à 9600 baud).
Une fois testé, vous pouvez alors passer à des exemples plus avancés.

L'exemple avancé **QTRAExample** (ou **QTRRCExample**) incorpore la calibration ainsi qu'une estimation de la position de la ligne.

Utilisation et notes

Sommaire

Référence des commandes QTRSensor

Les précédentes versions de la bibliothèque s'appelaient **PololuQTRSensors** mais il a été modifié en **QTRSensors** pour la différencier de la bibliothèque senseur QTR Arduino pour le Robot Orangutan et 3pi https://www.pololu.com/docs/0J17 . A part ce changement de nom de bibliothèque (et de classe), la nouvelle bibliothèque QTRSensors est fonctionnellement identique aux précédentes versions.

Pour les senseurs analogiques QTR-xA, vous aurez besoin d'instancier un objet **QTRSensorsAnalog** et pour les senseurs digitaux QTR-xRC vous aurez besoin d'instancier un objet **QTRSensorsRC**. A part les constructeurs, ces deux objets fournissent les mêmes méthodes pour lire les valeurs du senseur (les deux classes dérivent la même classe abstraite de base). Le bibliothèque offre un accès aux valeurs brutes du senseur ainsi que des fonctions de haut niveau incluant la calibration et suivit de ligne.

Cette section de la bibliothèque définit un objet pour chacun des deux types de senseur QTR avec la classe **QTRSensorsAnalog** destinée aux senseurs QTR-xA et une classe **QTRSensorsRC** destinée au senseurs QTR-xRC. En interne, la bibliothèque prend en charge les différences entre les QTR-xA et QTR-xRC offrant ainsi une interface commune pour les deux senseurs. La seule différence visible depuis l'extérieur, c'est le constructeur. Cela est possible car les deux classes dérive d'une classe commune **QTRSensors**, celle-ci offre une interface abstraite qui doit être implémentée dans les classe dérivées. La classe abstraite **QTRSensors** ne doit pas être instanciée.

Les classes QTRSensorsAnalog et QTRSensorsRC doivent être instanciées avant d'être utilisée. Cela permet à plusieurs senseurs de ligne QTR d*être contrôlé indépendamment les uns des autres.*

Pour la calibration, la mémoire est allouée en utilisant la fonction `malloc()`. Cela préserve la RAM: si les 8 senseurs sont calibrées avec l'émetteur activé et désactivé, un total de 64 octets (sur les 2048 disponibles) sont dédiés au stockage de la calibration. Cependant, pour une application utilisant uniquement 3 senseurs avec des émetteurs toujours actifs durant la lecteur alors seuls 6 octets seront nécessaires.

En interne, la bibliothèque utilise toutes des fonctions standards comme `micros()` pour la gestion du temps et `analogRead()` ou `digitalRead()` pour obtenir les valeurs du senseur. De sorte, cette bibliothèque devrait fonctionner avec tous les Arduino sans conflit avec d'autres bibliothèques.

Les fonctions

read()

```
void read(unsigned int *sensorValues, unsigned char readMode = QTR_EMITTERS_ON)
```

Lit les valeurs brutes des senseurs dans un tableau (*array*). Le tableau **DOIT** avoir la taille correspondant aux valeurs des senseurs spécifiés dans le constructeur. Les valeurs retournées sont une mesure de la réflectance (facteur de réflexion) en unités qui dépend du type de senseur

utilisé. Une valeur plus élevée indique une réflectance inférieure (une surface noire ou le vide). Les senseurs QTR-xA retournera une valeur brute entre 0 et 1023. Les senseurs QTR-xRC retournera une valeur entre 0 et un argument *timeout* (exprimé en microsecondes, indiqué dans le constructeur avec 2000 par défaut).

Toutes les fonction qui lisent des valeurs prennent un argument `readMode` en paramètre. Celui-ci spécifie le type de lecture qui est effectué. Plusieurs options sont possibles:

- QTR_EMITTERS_OFF indique que la lecture doit être effectuée sans allumer les LEDs infrarouge (IR), ce qui permet d'évaluer le niveau de lumière ambiante près du senseur;
- QTR_EMITTERS_ON indique que les diodes émettrices doivent être activées durant la lecture, ce qui permet de lire la réflectance;
- QTR_EMITTERS_ON_AND_OFF indique que la lecture doit être réaliser avec les deux états (allumé et éteint). La résultat retourné par l'option QTR_EMITTERS_ON_AND_OFF correspond à **allumé + max – éteint**, où "allumé" correspond à une lecture avec les LEDs IR allumée, "éteint" correspond à la lecture avec les LED IR éteinte et "max" la valeur maximale lue par le senseur.
  Cette option permet de réduire la quantité d'interférence provenant des variations de lumière ambiante.

A noter que le contrôle des LEDs émettrices ne fonctionnera que si vous avez spécifié une broche valide pour le contrôle des LEDs IR dans l'appel du constructeur.

Exemple d'utilisation:

```
unsigned int sensor_values[8];
sensors.read(sensor_values);
```

emittersOn()

```
void emittersOn()
```

Allume les LEDs IR (InfraRouge). Principalement utilisée par la méthode `read()` et appeler cette fonction avant ou après la lecture du senseur n'a aucun effet sur la lecture. Vous devriez utiliser cette fonction uniquement pour effectuer des tests.

Cette fonction en fait quelque-chose que si la broche émetteur IR (*emitter*) à été spécifiée dans le constructeur (donc une valeur différente de QTR_NO_EMITTER_PIN).

emittersOff()

```
void emittersOff()
```

Désactive les LEDs Infrarouges. Comme pour `emitterOn()`, cette fonction est utilisée par `read()` et appeler cette fonction avant ou après une lecture de senseur n'a aucun effet sur les lectures (mais peut être utilisée pour effectuer des tests).

calibrate()

```
void calibrate(unsigned char readMode = QTR_EMITTERS_ON)
```

Effectue une lecture des senseurs pour calibration. Les valeurs du senseur ne sont pas retournée; à la place, les valeurs maximales et minimales sont stockées en interne et utilisé par la méthode **readCalibrated()** (*lecture calibrée*). Vous pouvez accéder aux valeurs de calibration (ex: les lectures min et max du senseur) via les pointeurs des membres publiques **calibratedMinimumOn**, **calibratedMaximumOn**, **calibratedMinimumOff** et **calibratedMaximumOff**. Notez que ces pointeurs pointent vers des tableaux ayant une longueur `numSensors` tel que spécifié dans le constructeur. Ces tableaux ne seront alloués qu'après l'appel de **calibrate()**. Si la la calibration est uniquement réalisée avec les LED IR allumées alors les tableaux de calibration destinés aux valeurs **off** (*éteinte*) ne seront pas alloués.

readCalibrated()

```
void readCalibrated(unsigned int *sensorValues, unsigned char readMode = QTR_EMITTERS_ON)
```

Retourne la lecture des senseurs calibrés avec une valeur entre 0 et 1000, où 0 corresponds à une lecture inférieure ou égale à la valeur minimale obtenue par **calibrate()** et 1000 correspond à une lecture supérieure ou égale à la valeur maximale. Les valeurs de calibration sont stockées séparément pour chacun des senseurs infrarouges, par conséquent, la différence entre les senseurs est automatiquement prise en compte.

readLine()

```
unsigned int readLine(unsigned int *sensorValues, unsigned char readMode = QTR_EMITTERS_ON, unsigned char whiteLine = 0)
```

Fonctionne de la même façon que `readCalibrated` mais cette fonction est conçue pour le suivit de ligne:

- Cette fonction retourne une estimation de la position de la ligne.
- L'estimation est faite en utilisant la moyenne pondérée des senseurs par indice, indice du senseur multiplié par 1000. De sorte, une valeur 0 indique que la ligne est sous le senseur 0 (ou qu'il a été sous le senseur 0 juste avant d'être perdu), une valeur de 1000 indique que le senseur est sous le senseur 1, 2000 indique sous le senseur 2, etc.

Une valeur intermédiaire indique indique que la ligne est entre deux senseurs. La formule est:

```
 0*value0 + 1000*value1 + 2000*value2 + ...
-------------------------------------------
     value0  +  value1  +  value2 + ...
```

Aussi longtemps que les senseurs ne sont pas trop espacés par rapport à la ligne alors cette fonction est conçue pour retourner une valeur monotone. Une fonction monotone reste toujours croissante ou décroissante (son sens de variation est constant, voir l'article Fonction monotone sur wikipedia.fr https://fr.wikipedia.org/wiki/Fonction_monotone).

Une valeur monotone est une entrée idéale pour réaliser un asservissement PID en boucle fermée.

De surcroît, cette méthode se souvient où elle a vu la ligne la dernière fois, de sorte que si la ligne est perdue sur la gauche ou la droite, la position de la ligne sera indiquée du bon côté afin de diriger le robot dans la bonne direction pour rattraper la ligne.

Par exemple, si le senseur 4 est le senseur le plus à droite et que vous quittez la ligne par la gauche (dont la ligne est quelque par plus loin sur la droite), la valeur lue augmente progressivement jusque 4000 puis continuera à retourner 4000 lorsque le robot aura quitté la ligne.

Par défaut, cette fonction par du principe que **la ligne est noir** (valeur élevée) entourée de blanc (valeur faible). Si vous désirez utiliser des lignes blanches sur fonds noir alors vous pouvez placer le second argument `whiteLine` à true. Dans ce cas, chaque senseur sera remplacé par la valeur maximale du senseur moins sa valeur actuelle (avant le calcul de moyenne).

calibratedMinimumOn

```
unsigned int* calibratedMinimumOn
```

Les valeurs minimales de calibration pour chaque senseur (avec diode IR allumée). Les pointeurs ne sont pas alloués (et placés à 0) jusqu'à ce la méthode **calibrate()** soit appelée. Ensuite le pointeur pointera vers un tableau qui aura exactement la taille du nombre de senseur. En fonction du `readMode` utiliser avec **calibrate()** seules les valeurs *On* ou *Off* seront allouées (en fonction des besoins).

Cette variable et les suivantes sont rendues publiques pour que vous puissiez réaliser vos propres calculs et réaliser des t^aches comme sauvegarde en EEPROM, réaliser un contrôle sur les valeurs, etc

calibratedMaximumOn

```
unsigned int* calibratedMaximumOn
```

Les valeurs maximales de calibration mesurée pour chaque senseur (avec les diode IR allumées).

calibratedMinimumOff

```
unsigned int* calibratedMinimumOff
```

Les valeurs minimales de calibration pour chaque senseur, avec les diodes émettrices eteintes.

calibratedMaximumOff

```
unsigned int* calibratedMaximumOff
```

Les valeurs de calibrations maximales mesurées pour chaque senseur (avec les diodes IR éteintes).

~QTRSensors() - descructeur

```
Destructor: ~QTRSensors()
```

Destructeur de la classe QTRSensors qui libère la mémoire allouée par les tableaux de calibration.

QTRSensorsRC() - constructeur

```
Constructor: QTRSensorsRC()
```

Cette version du constructeur n'effectue aucune initialisation. Si ce constructeur est utiliser alors le code doit également appeler `init()` avant d'utiliser les méthodes de la classe.

```
Constructor: QTRSensorsRC(unsigned char* digitalPins, unsigned char numSensors, unsigned int timeout = 2000, unsigned char emitt
```

Cette version du constructeur appelle la méthode `init()` décrite ci-dessous.

```
void QTRSensorsRC::init(unsigned char* digitalPins, unsigned char numSensors, unsigned int timeout = 2000, unsigned char emitter
```

Initialise le tableau de senseur QTR-RC (digital).

- Le tableau *digitalPins* doit contenir les broches digitales Arduino correspondant à chaque senseur.
- *numSensors* spécifie la longueur du tableau *digitalPins* (le nombre de senseur QTR-RC que vous utilisez). *numSensors* ne doit pas excéder 16 positions.
- *timeout* spécifie le nombre de microsecondes au-delà desquels la lecture du senseur sera considérée comme complètement noir! Cela signifie, si la longueur de l'impulsion pour une broche excède le *timeout*, la mesure de l'impulsion cesse et la lecture pour cette broche considère que le résultat est complètement noir. Pololu recommande un timeout entre 1000 et 3000 µs (0.001 et 0.003 secondes) dépendant de facteurs comme la hauteur du senseur et la lumière ambiante. Cela permet d'écourter le cycle de lecture des senseurs tout en maintenant une mesure utile de la reflectande.
- *emitterPin* est la broche digitale Arduino qui contrôle l'allumage et l'extinction des LEDs Infrarouge. Cette broche est optionnelle, broche qui n'existe que sur les détecteurs de lignes QTR 8A et 8RC QTR. Si une broche valide est spécifié alors les LED InfraRouges ne sont allumée que durant les opérations de lecture. Si la valeur QTR_NO_EMITTER_PIN (255) est utilisé lors de l'appel alors vous pouvez laisser le broche *emitter* déconnectée sur votre senseur de ligne afin que les LED Infrarouge restent toujours allumées.

QTRSensorsAnalog() - constructeur

```
Constructor: QTRSensorsAnalog()
```

Cette version du constructeur -destiné au senseurs de type analogique- n'effectue aucune initialisation. Si ce constructeur est utilisé, l'utilisateur doit appeler la méthode init() avant l'utilisation des autres méthodes de cette classe.

```
Constructor: QTRSensorsAnalog(unsigned char* analogPins, unsigned char numSensors, unsigned char numSamplesPerSensor = 4, unsign
```

Ce constructeur fait juste un appel à `init()`, tel que détaillé ci-dessous.

```
void init(unsigned char* analogPins, unsigned char numSensors, unsigned char numSamplesPerSensor = 4, unsigned char emitterPin =
```

Initialise un ensemble de senseur QTR-A (analogique).

- **pins** est un tableau de broches qui contient les broches d'entrées analogiques utilisées pour chaque senseur. Par exemple, si les pins sont {0, 1, 7}, le premier senseur est l'entrée analogique 0, le deuxième senseur est l'entrée analogique 1 et le troisième senser est l'entrée analogique 7.
- **numSensors** indique la longueur du tableau *analogPins* (le nombre de senseurs QTR-A utilisés). La valeur de *numSensors* ne peut pas dépasser 16.
- **numSamplesPerSensor** indique le nombre d'échantillonnages 10-bit à réaliser par canal/senseur lors de chaque lecture (échantillons dont la bibliothèque fait une moyenne). Le nombre total de conversion analogique-vers-digital réalisés est également au numSensors * numSamplesPerSensor.
  accroître ce paramètre améliore la suppression du bruit mais au coût d'un temps de capture plus important (accroissement du temps d'échantillonnage total). Ce paramètre ne peut pas excéder la valeur de 64. Pololu recommande la valeur 4.
- **emitterPin** est la broche digitale Arduino qui permet de contrôler les LEDs infrarouges (allumées ou éteintes). Cette broches est optionnelle et elle n'existe que sur les modèles de QTR 8A et QTR 8RC des senseurs de ligne. Si une broche valide est spécifiée, alors les diodes infrarouge ne seront allumées que durant la lecture. Si la valeur `QTR_NO_EMITTER_PIN` (255) est utilisée pour ce paramètre alors vous pouvez laisser la broches des diode Infrarouge déconnectée et les LEDs seront constamment alimentées.

Note d'usage

Calibration

La bibliothèque propose la méthode **calibrate()** pour facilement calibrer les senseurs dans les conditions particulières qu'il va rencontrer. La

calibration des senseurs peu permet de produire des lectures sensiblement plus fiable, ce qui aura pour effet de simplifier votre propre code par la suite. En conséquence, nous recommandons de réaliser une phase de calibration dans la routine d'initialisation de votre application.

Ce peu simplement être réaliser durant une période de temps fixe durant laquelle vous effectuer des appels récurrents à la méthode **calibrate()**. Durant cette phase de calibration, vous devrez exposer chaque senseur de réflectance aux lectures les plus claires et foncées auxquels il seront exposés. Par exemple, si vous réalisez un suiveur de ligne, vous aurez besoin de le faire glisser progressivement au dessus de la ligne durant la phase de calibration de sorte que chaque senseur puisse avoir l'occasion de faire une lecture sur la surface la plus réflective (la plus claire) et de la surface la moins réfléchissante (la plus sombre).

Un exemple de routine de calibration peut être:

```
#include <QTRSensors.h>

// créer un objet pour votre type de senseur (RC ou Analogique)
// Dans cet exemple, il s'agit de trois senseur sur les
// entrées analogiques 0 à 2 (donc les broches digitales de
// 14 à 16)
QTRSensorsRC qtr((char[]) {14, 15, 16}, 3);
// QTRSensorsA qtr((char[]) {0, 1, 2}, 3);

void setup()
{
  // optionnel: attendre une action de l'utilisateur comme
  // la pression d'un bouton.

  // Puis démarrer la phase de calibration et déplacez le
  // senseurs au dessus des deux surfaces avec les reflectances
  // opposées (extrêmes) que votre application rencontrera:
  int i;
  for (i = 0; i < 250; i++)  // Réaliser une pendant ~5 secondes
  {
    qtr.calibrate();
    delay(20);
  }

  // optionnel: signaler que la phase de calibration est
  // achevée et attendre une action de l'utilisateur
  // (comme par exemple, la pression d'un bouton)
}
```

Lectures avec les senseurs

Cette bibliothèque offre différentes approches pour lire les senseurs:

1. Vous pouvez obtenir les valeurs brutes des senseurs en utilisant la méthode **read()**, qui reçoit un argument optionnel qui vous permet d'effectuer une lecture avec les diodes émettrices éteintes (note: l'extinction des diodes émettrices est uniquement possible pour les senseurs QTR-8x).
2. Vous pouvez obtenir les valeurs calibrées en utilisant la méthode **readCalibrated()**, qui dispose également d'un argument optionnel vous permetant de faire une lecture avec les LED IR éteintes. Les valeurs calibrées ont toujours une valeur située entre 0 à 1000, où 0 étant considérée comme la plus réfléchissante (ex: la plus blanche) que la surface la plus réfléchissante rencontrée durant la phase de calibration -ET- 1000 étant la surtface la moins réfléchissante (ex: plus noir) que la surface la moins réfléchissante rencontrée durant la phase de calibration.
3. Les applications de détection de ligne peuvent demander la position de la ligne en utilisant la méthode **readLine()**. **readline()** accepte un premier argument booléen optionnel qui indique la couleur de la ligne (ligne noire sur fond blanc -ou- ligne blanche sur fond noir). Le second argument optionnel indique si les LEDs Infrarouge doivent être allumées ou éteintes durant la mesure. **readLine()** utilise les valeurs de calibration pour chaque senseur et retourne un entier qui indique la position supposée de la ligne. Si vous utilisez N senseurs, une valeur 0 retournée signifie que la ligne est sous le senseur 0 (ou au-delà du senseur 0) et retourne une valeur 1000x(N-1) si la ligne est sous le senseur N-1 (ou entre le senseur N-1 -dernier senseur) ou au-delà de ce dernier).

Lorsque vous déplacez le senseur perpendiculairement à la ligne, la position de la ligne changera de façon monotone entre 0 et 1000 * (N-1) ou vice versa. Changer de façon monotone indique le la valeur croit (ou décroit) constamment dans le même sens.

Cette position de ligne peut être utilisée pour dans un contrôle PID en boucle fermée.

Une routine élémentaire permettant de suivre une ligne en utilisant les valeurs du senseur et effectuer le suivit de ligne pourrait ressembler à ceci:

```
void loop()
{
  unsigned int sensors[3];
  // obtenir les valeurs calibrées du senseur (dans un tableau de senseur)
  // ainsi que la position de la ligne dans une gamme de valeur de
  // 0 à 2000, avec 1000 retournée pour la ligne au milieu du senseur.
  int position = qtr.readLine(sensors);

  // Si tous les senseurs ont une très faible réflectance, alors prendre
  // un action appropriée pour cette situation.
  if (sensors[0] > 750 && sensors[1] > 750 && sensors[2] > 750)
```

```
  {
    // Faire une action. Cela peu signifier que l'on est à la fin
    // de la course ou peut être tombé de la table, auquel cas, nous
    // nous devrions arrêter de bouger, se retourner, aller à la
    // recherche de la ligne.
    return;
  }

  // Calculer l' "erreur" par rapport à la position de la ligne.
  // Nous allons faire en sorte que l'erreur = 0 lorsque la ligne est
  // placée sous le milieu du senseur (parce que c'est notre but).
  // L'erreur aura une valeur entre -1000 et +1000.
  // Si nous avons le senseur 0 à gauche et le senseur 2 à droite alors
  // une lecture d'erreur de -1000 signifie que nous voyons la ligne
  //  sur la gauche par rapport au centre du senseur alors qu'une
  // lecture de +1000 signifie que la ligne est sur la droite
  // par rapport au centre.
  int error = position - 1000;

  // Vitesse des moteurs
  int leftMotorSpeed = 100; // gauche
  int rightMotorSpeed = 100;// droite
  if (error < -500)  // ligne sur la gauche
    // tourner à droite
    // -> arrêter moteur gauche
    leftMotorSpeed = 0;
  if (error > 500)  // ligne sur la droite
    // tourner à gauche
    // -> arrêter le moteur droit
    rightMotorSpeed = 0;

  // Fixer la vitesse des moteurs en utilisant
  // les valeurs de leftMotorSpeed et rightMotorSpeed
}
```

Contrôle PID

TLa valeur entière retournée par **readLine()** peu facilement être convertie la mesure de notre erreur de position pour les applications de suivit de ligne, comme démontré dans le précédent exemple de code. La fonction utilisée pour générer cette valeur de position/error est conçue pour être monotonique, ce qui signifie que la valeur change toujours dans la même direction au fur et a mesure que que les senseurs croisent la ligne. Cela en fait une grandeur utile pour réaliser un contrôle PID (*Proportionnel Intégram Dérivé*).

Expliquer la nature du contrôle PID va au delà des objectifs de ce document, mais wikipedia offre un bon article https://fr.wikipedia.org/wiki/R%C3%A9gulateur_PID sur le sujet.

Le code suivant est un exemple très simple de contrôleur PD pour suivit de ligne (le terme intégrale "I" d'un régulateur PID n'est habituellement pas nécessaire dans le cadre d'un suivit de ligne). Les valeurs des différentes constantes PID (ou PD dans le cas présent) est généralement spécifique à votre propre application mais d'une façon générale vous pourriez noter que la constante de dérivation *Kd* est habituellement beaucoup plus grande que la constante proportionnelle *Kp*. C'est parce que la dérivée de l'erreur (quantification de la *variation de l'erreur*) est beaucoup plus petite que l'erreur elle même. Par conséquent, pour produire une correction signification il est nécessaire de multiplier le terme dérivé par une constante beaucoup plus grande.

```
int lastError = 0;

void loop()
{
  unsigned int sensors[3];
  // Obtenir les valeurs calibrées retournées dans
  // tableau de senseur accompagné de la position
  // de la ligne (valeur entre 0 et 2000) avec
  // 1000 correspondant à la ligne au milieu du
  // senseur
  int position = qtr.readLine(sensors);

  // Calculer notre "erreur" depuis la position
  // de la ligne. Nous allons la calculer de sorte
  // que l'erreur égale 0 si la ligne est sous la
  // position centrale du senseur (ce qui est notre
  // but). L'erreur aura une valeur entre -1000 et
  // +1000. Si nous avons le senseur 0 à gauche et
  // le senseur 2 à droite, alors une valeur lue
  // de -1000 signifie que la ligne est vue sur la
  // gauche du senseur. Une valeur de +1000
  // signifie que que nous voyons la ligne sur
  // la droite du senseur.
  int error = position - 1000;

  // Baser la vitesse des moteurs sur base des
  // termes proportionnel et dérivés d'un PID.
  // KP est la constante proportionnelle (commencer
  //    avec une valeur 0.1)
  // KD est la constance dérivée (vous pouvez peut-être
  //    commencer avec un valeur autour de 5)
  // Note: lors d'un asservissement PID, il est très
  //   important d'avoir le bon signe pour les valeurs
```

```
  //   sinon, la boucle de contrôle sera totalement
  //   instable.
  // motorSpeed peut avoir une valeur positive ou
  //   négative en fonction du sens de la direction
  //   prise.
  int motorSpeed = KP * error + KD * (error - lastError);
  lastError = error;

  // M1 et M2 sont les vitesses par défaut des moteurs.
  // Ce sont les vitesses adéquate des deux moteurs
  //    pour suivre -SANS ERREUR- une ligne parfaitement
  //    droite.
  // Si vos moteurs sont identiques alors M1 et M2 devraient
  //    être égale (ou presque, chaque moteur étant un peu
  //    différent de l'autre).
  // Il est préférable de commencer avec une petite valeur
  //    pour M1 et M2 lorsque vous commencez à tester votre
  //    boucle de contrôle. En effet, l'erreur (écart par
  //    rapport à la ligne) devient rapidement plus grande
  //    si la vitesse est plus élevée.
  //    Après avoir atteint un résultat satisfaisant à
  //    faible vitesse, vous pouvez augmenter les valeurs
  //    de M1 et M2 et poursuivre l'affinement des valeurs
  //    des constantes KP et KD de votre PID.
  int m1Speed = M1 + motorSpeed;
  int m2Speed = M2 - motorSpeed;

  // Cela peut aider de maintenir une vitesse positive
  // pour les moteurs (ce point est optionnel).
  // Un test similaire pourrait être ajouté pour garder
  // la vitesse des moteurs en dessous du maximum
  // autorisé.
  if (m1Speed < 0)
    m1Speed = 0;
  if (m2Speed < 0)
    m2Speed = 0;

  // Fixer la vitesse des mmoteur en utilisant les
  // variables m1Speed et m2Speed
}
```
